

## **REMARKS/ARGUMENTS**

Claims 1-15 are pending in the present application. Claims 6, 9, 11-12, and 14-15 were amended; claims 1-5 were canceled; claims 16-19 were added. No new matter has been added. Support for the new claims can be found in the specification on page 22, lines 8-26; page 23 line 8, through page 25, line 4; page 27, lines 1-5; and page 34, lines 6-30. Reconsideration of the claims is respectfully requested.

### **I. 35 U.S.C. § 101**

The Examiner has rejected claims 6-15 under 35 U.S.C. § 101 as being directed towards non-statutory subject matter.

The Examiner states:

**Claims 6-10** are directed to "A data processing system ... comprising:" means for performing various functionalities. These means are not necessarily associated with any hardware aspect of a data processing system, thus represent only software per se.

**Claims 11-15** are not limited to tangible embodiments. In view of Applicant's disclosure, specification page 64 in the last paragraph, the medium is not limited to tangible embodiments, instead being defined as including both tangible embodiments (e.g., "recordable-type media, such as a floppy disk") and intangible embodiments (e.g., transmission-type media, such as ... radio frequency and light wave transmissions"). As such, the claim is not limited to statutory subject matter and is therefore non-statutory. Office Action dated May 14, 2007, pages 2-3.

Applicants have amended claim 6 to describe a data processing system comprising: an instruction cache, which is included in a processor. Therefore, claims 6-10 do not represent software.

Applicants have amended claim 11 to describe a computer program product, which is stored in a computer readable medium. Therefore, claims 11-15 are limited to tangible embodiments.

Therefore the rejection of claims 6-15 under 35 U.S.C. § 101 has been overcome.

### **II. 35 U.S.C. § 103, Obviousness**

The Examiner has rejected claims 1-15 under 35 U.S.C. § 103(a) as being unpatentable over *Lueh*, Static Compilation of Instrumentation Code for Debugging Support, U.S. Patent No. 6,966,057, dated November 15, 2005 (hereinafter referred to as "*Lueh*") in view of U.S. Patent Application Publication 2003/0225917, published by *Partamian*. This rejection is respectfully traversed.

Applicants' claims 6 and 11 recite similar features. Claim 6 describes each one of a plurality of individual instructions associated with an indicator that indicates that each one of the plurality of instructions needs to be monitored. The indicator is stored in at least one existing spare bit in each one of the plurality of individual instructions. An instruction cache uses said indicator to detect execution of each one of the plurality of instructions, wherein execution of instructions, which are not associated with

the indicator, is not detected. The combination of *Lueh* and *Partamian* does not render Applicants' claims obvious because the combination does not teach or suggest these features.

The Examiner states:

**Regarding Claims 1, 6 and 11:** Lueh discloses profiling an application in a data processing system, the method comprising: detecting execution of an instruction associated with an indicator, wherein the instruction is located in a routine (col. 7, lines 9-16 "instrumentation code passes necessary information to a run-time library function"); determining whether the instruction is 'hot' (col. 4, lines 60-65 "methods that are identified as hot methods based on the collected profiling information"); and responsive to the instruction having been identified as 'hot', generating an interrupt to pass control to a monitoring program (col. 9, lines 14-35 "The execution 640 includes . . . executing an event hook function for an event corresponding to the field watch"; col. 7, lines 5-8 "To support the watch points for fields, the JIT compiler interrupts the execution ... to call the event hook function"), wherein the monitoring program identifies information regarding a caller of a routine (col. 5, lines 11-16 "the JIT compiler provides a mechanism to identify and access the caller's frame context, referred to as unwinding stack frame.").

Lueh does not explicitly disclose determining whether the instruction is 'hot' comprises determining whether the instruction has been executed more often than a threshold value.

Partamian teaches method of determining whether an instruction is 'hot' comprising determining whether the instruction has been executed more often than a threshold value (par. [0018] "JVM 1120 includes a threshold to determine whether a method is hot or not.")

Office Action dated May 14, 2007, pages 3-4.

*Lueh* teaches JAVA Virtual Machine Debug Interface (JVMDI) field watches. Field access events are generated when a specified field is about to be accessed.

The basic support for field watch is to insert an instrumentation code to the method's code space. This instrumentation code passes necessary information to a run-time library function, which calls the event hook function. To prevent significant runtime overhead caused by un-activated field watches, the execution of the instrumentation code is guarded by modifying the method's address space or using Boolean flag for each Java class field.

*Lueh*, column 7, lines 8-15.

*Lueh* teaches a field access and modification watch 445 which has three models, i.e. static, semi-static, and dynamic, to use to insert the additional instrumentation code.

The three models, or approaches, differ by where and when the additional instrumentation code is inserted and how the execution of the instrumentation code is guarded during debugging support. In the static model, the additional code is inserted in method's address space at compilation time and its execution is guarded using a Boolean flag. In the semi-static model, the additional code is inserted at the end of the method's code space and/or a no-op (NOP) instruction sequence is inserted to the location that access or modify fields. The execution of the

instrumentation code is guarded by changing NOP instruction sequence to a jump instruction. In the dynamic model, dynamic recompilation is used to insert the instrumentation code when needed. All three models or use the same mechanism to pass the arguments to and call the run-time library function that calls the event hook function.

*Lueh*, column 7, lines 34-50.

Figures 7-9 depict examples for these three models. For example, Figure 7 depicts an example for the static model. For the static model, the instrumentation code is inserted before a field access or modification point. Lines 3-5 of the depicted code show the native instruction sequence. The instrumentation code, which was inserted to the native instruction sequence, is lines 8-10.

Regarding Applicants' original claims, the Examiner relies on *Lueh*, column 7, lines 9-16, to teach detecting execution of an instruction associated with an indicator wherein the instruction is located in a routine. Specifically, the Examiner refers to instrumentation code that passes necessary information to a run-time library function.

The combination of *Lueh* and *Partamian* does not render Applicants' claims obvious because neither *Lueh* nor *Partamian* teaches or suggests each one of a plurality of individual instructions associated with an indicator that indicates that each one of the plurality of instructions needs to be monitored, the indicator stored in at least one existing spare bit in each one of the plurality of individual instructions.

*Lueh* teaches inserting instrumentation code in a method's code. The instrumentation code supports field watch. The instrumentation code is not an indicator that is stored in at least one existing spare bit in an individual instruction that needs to be monitored.

The combination of *Lueh* and *Partamian* also does not render Applicants' claims obvious because neither *Lueh* nor *Partamian* teaches or suggests using the indicator to detect execution of each one of the plurality of instructions, wherein execution of instructions, which are not associated with the indicator, is not detected. The instrumentation code of *Lueh* does not detect execution of instructions. The instrumentation code is itself executed under certain conditions. *Lueh* does not teach execution of instructions, which are not associated with the indicator, not being detected.

*Partamian* does not cure the deficiencies of *Lueh*. *Partamian* teaches analyzing the results of the execution of a Java program, but does not teach each one of a plurality of individual instructions associated with an indicator that indicates that each one of the plurality of instructions needs to be monitored. In the present application, the indicator is stored in at least one existing spare bit in each one of the plurality of individual instructions. An instruction cache uses said indicator to detect execution of each one of the plurality of instructions, wherein execution of instructions, which are not associated with

the indicator, is not detected. Therefore, the combination of *Lueh* and *Partamian* does not render Applicants' claims obvious.

Dependent claims 7-10 and 12-18 depend from one of the independent claims discussed above and are patentable at least by virtual of their dependency.

Therefore, the rejection of claims 1-15 under 35 U.S.C. § 103(a) has been overcome.

### **III. Conclusion**

It is respectfully urged that the subject application is patentable over the cited prior art and is now in condition for allowance.

The examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: August 14, 2007

Respectfully submitted,

/Lisa L.B. Yociss/

Lisa L.B. Yociss  
Reg. No. 36,975  
Yee & Associates, P.C.  
P.O. Box 802333  
Dallas, TX 75380  
(972) 385-8777  
Attorney for Applicants